FIG.1

STATE MACHINE GENERATION

REDUCTION IN NUMBER OF STATES	S5A
OPTIMIZATION OF CODE	S5B
RETAIN ANALYSIS	S5C
STATE MACHINE EXTRACTION	S5D

S5

STATE MACHINE STORAGE SECTION — 7

HDL DESCRIPTION GENERATION — S6

HDL DESCRIPTIONS — 4

COMPILER PROCESSING — 2

PSEUDO C DESCRIPTIONS (PSEUDO C PROGRAM) — 1

LOAD PROGRAM — S1

IDENTIFY REGISTER ASSIGNMENT STATEMENT — S2

REGISTER INFORMATION STORAGE SECTION — 5

CFG GENERATION — S3

INTERMEDIATE REPRESENTATION STORAGE SECTION — 6

C DESCRIPTION GENERATION — S4

C DESCRIPTIONS (C PROGRAM) — 3

FIG.2

valid_a ——→

valid_b ——→

a[14:0] ——→

b[14:0] ——→

10

PIPELINE ADDITION CIRCUIT
WHICH IS ATTENDED
WITH STALL OPERATION

——→ valid_out

——→ out[15:0]

FIG.3

clk

valid_a

a[14:0]   a1   a2   a3

valid_b

b[14:0]   b1   b2   b3

valid_out

out[15:0]   a1 + b1   a2 + b2   a3 + b3

out[15:0] = a[14:0] + b[14:0]

FIG.4
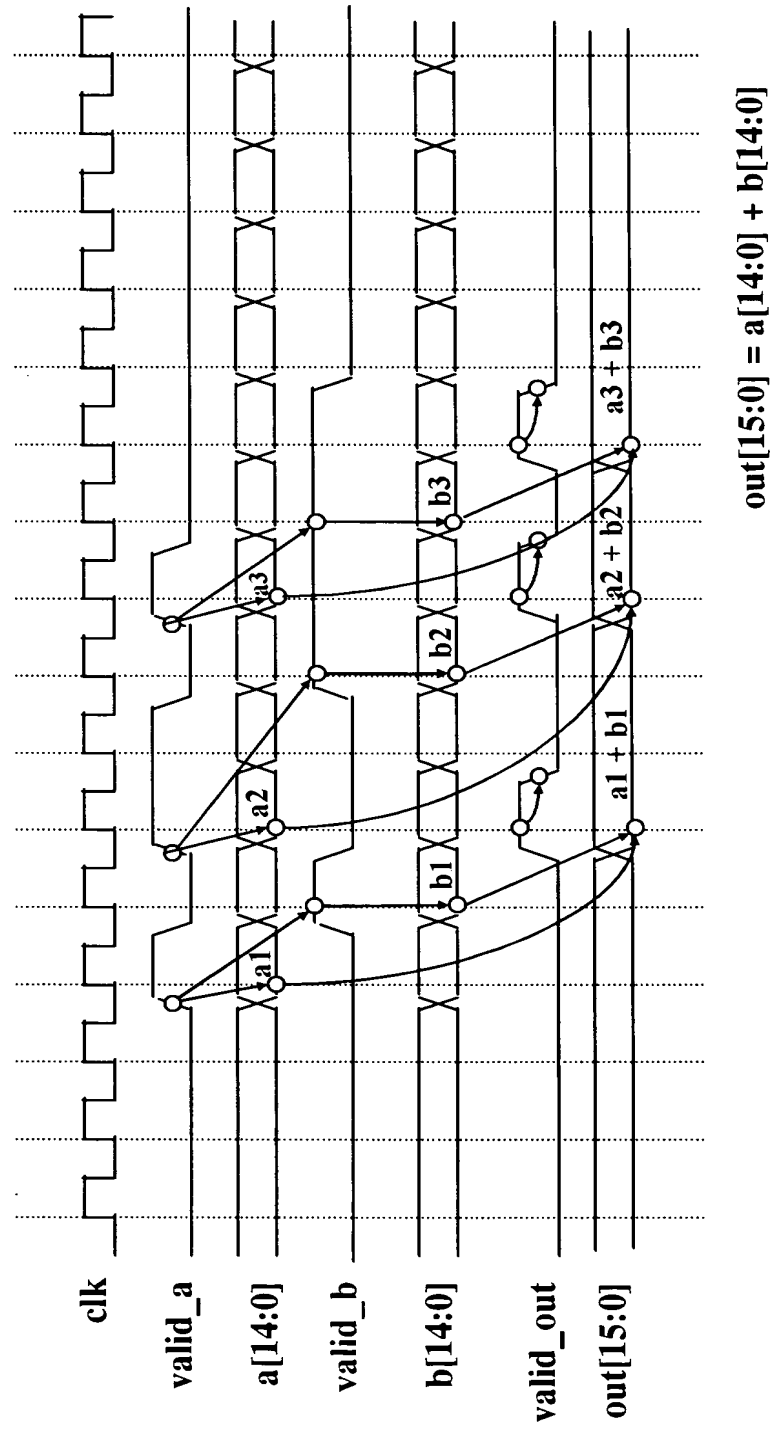
```
1  #include <stdio.h>
2  void pipeline(unsigned short valid_a,
3                 unsigned short valid_b,
4                 unsigned short a,
5                 unsigned short b,
6                 unsigned short *out,
7                 unsigned short *valid_out);
8  main() {
9    unsigned short valid_a, valid_b,
10                  a, b, *out, *valid_out;
11   *out      = 0x0000;
12   *valid_out = 0x0000;
13   pipeline(valid_a, valid_b, a, b, out, valid_out);
14 }
```

```
15 void pipeline(unsigned short valid_a, unsigned short valid_b,
16               unsigned short a, unsigned short b,
17               unsigned short *out, unsigned short *valid_out) {
18 unsigned short valid_a_tmp = 0x0000;
19 unsigned short a_tmp = 0x0000;
20 unsigned short b_tmp = 0x0000;
21 while (1) {
22   valid_a_tmp = $ 0x0001&valid_a;
23   if ((0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)) {
24     a_tmp = 0x7FFF&a;
25     $
26 L :
27     if (0x0001&valid_b == 0x0001) b_tmp = 0x7FFF&b;
28     else                $ goto L;
29     *out = $ (a_tmp + b_tmp);
30     *valid_out = $ 0x0001;
31   } else {
32     $
33     *valid_out = $ 0x0000;
34   }
35 }
36 }
```

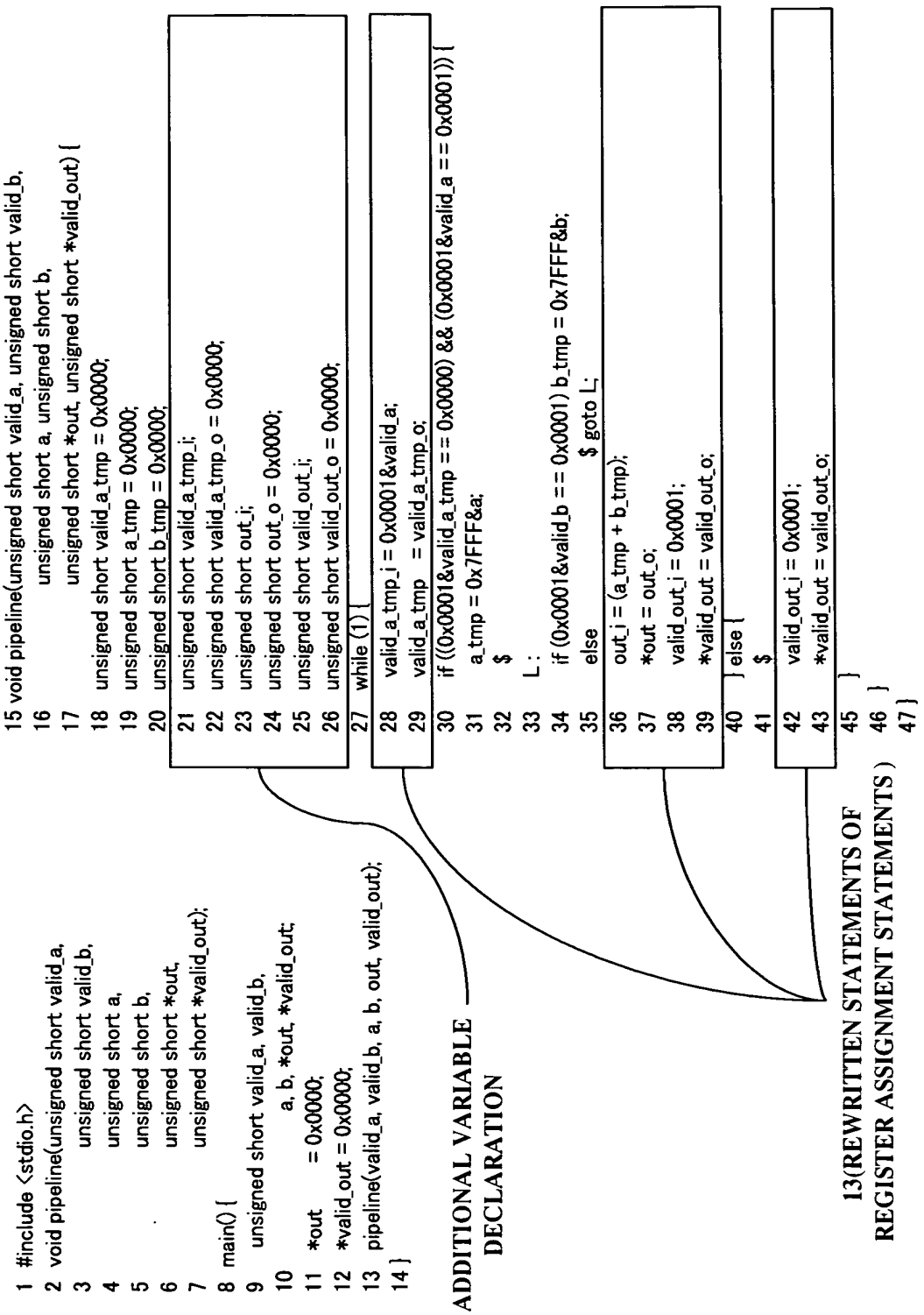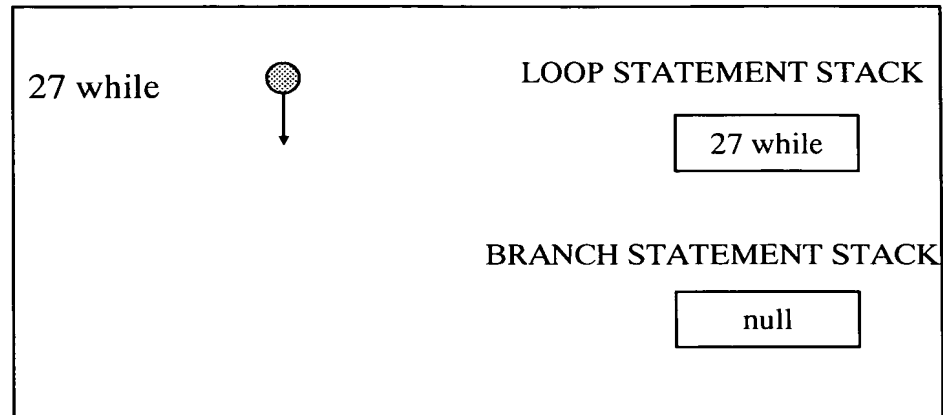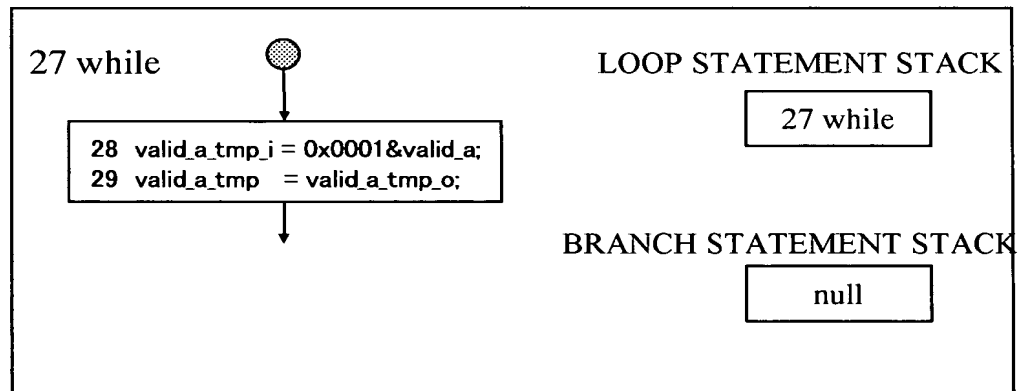11 (CIRCUIT OPERATION DESCRIPTION PART)

FIG.5

```
1  #include <stdio.h>
2  void pipeline(unsigned short valid_a,
3          unsigned short valid_b,
4          unsigned short a,
5          unsigned short b,
6          unsigned short *out,
7          unsigned short *valid_out);
8  main() {
9    unsigned short valid_a, valid_b,
10          a, b, *out, *valid_out;
11   *out    = 0x0000;
12   *valid_out = 0x0000;
13   pipeline(valid_a, valid_b, a, b, out, valid_out);
14 }
```

ADDITIONAL VARIABLE
DECLARATION

```
15  void pipeline(unsigned short valid_a, unsigned short valid_b,
16          unsigned short a, unsigned short b,
17          unsigned short *out, unsigned short *valid_out) {
18    unsigned short valid_a_tmp = 0x0000;
19    unsigned short a_tmp = 0x0000;
20    unsigned short b_tmp = 0x0000;
21    unsigned short valid_a_tmp_i;
22    unsigned short valid_a_tmp_o = 0x0000;
23    unsigned short out_i;
24    unsigned short out_o = 0x0000;
25    unsigned short valid_out_i;
26    unsigned short valid_out_o = 0x0000;
27    while (1) {
28      valid_a_tmp_i = 0x0001&valid_a;
29      valid_a_tmp  = valid_a_tmp_o;
30      if ((0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)) {
31        a_tmp = 0x7FFF&a;
32        $
33  L:
34        if (0x0001&valid_b == 0x0001) b_tmp = 0x7FFF&b;
35        else        $ goto L;
36        out_i = (a_tmp + b_tmp);
37        *out = out_o;
38        valid_out_i = 0x0001;
39        *valid_out = valid_out_o;
40      } else {
41        $
42        valid_out_i = 0x0001;
43        *valid_out = valid_out_o;
45      }
46    }
47 }
```

13(REWRITTEN STATEMENTS OF
REGISTER ASSIGNMENT STATEMENTS)

FIG.6

27 while

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

null

FIG.7

27 while

```
28  valid_a_tmp_i = 0x0001&valid_a;
29  valid_a_tmp   = valid_a_tmp_o;
```

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

null

FIG.8

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;

c1 30 if

~c1

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

30 if

$$C1 = (0x0001 \& valid\_a\_tmp == 0x0000) \&\& (0x0001 \& valid\_a == 0x0001)$$

FIG.9

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;

c1 30 if

31 a_tmp = 0x7FFF&a;

~c1

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

30 if

$$C1 = (0x0001 \& valid\_a\_tmp == 0x0000) \&\& (0x0001 \& valid\_a == 0x0001)$$

FIG.10

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

c1   30 if

31 a_tmp = 0x7FFF&a;

32 $

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

30 if

~c1

$$C1 = (0x0001\&valid\_a\_tmp == 0x0000) \&\& (0x0001\&valid\_a == 0x0001)$$

FIG.11

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

.c1   30 if

31 a_tmp = 0x7FFF&a;

32 $

33 L

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

30 if

~c1

$$C1 = (0x0001\&valid\_a\_tmp == 0x0000) \&\& (0x0001\&valid\_a == 0x0001)$$

FIG.12

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

c1   30 if

31 a_tmp = 0x7FFF&a;

32 $

33 L

c2   34 if
¬c2

¬c1

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

| 34 if | 30 if |

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.13

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

30 if

~c1

c1

31 a_tmp = 0x7FFF&a;

32 $

33 L

34 if

c2

~c2

34 b_tmp = 0x7FFF&b

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

34 if	30 if

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.14

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

34 if	30 if

27 while

| 28 valid_a_tmp.i = 0x0001&valid_a; |
| 29 valid_a_tmp   = valid_a_tmp_o; |

30 if

c1

~c1

| 31 a_tmp = 0x7FFF&a; |

32 $

33 L

c2

34 if

~c2

35 $

| 34 b_tmp = 0x7FFF&b |

$$C1 = (0x0001\&valid\_a\_tmp == 0x0000) \ \&\& \ (0x0001\&valid\_a == 0x0001)$$

$$C2 = 0x0001\&valid\_b == 0x0001$$

FIG.15

27 while

28 valid_a_tmp.i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

30 if

~c1

c1

31 a_tmp = 0x7FFF&a;

32 $

33 L

34 if

~c2

c2

35 $

34 b_tmp = 0x7FFF&b

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

34 if	30 if

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.16

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

30 if

27 while

| 28 valid_a_tmp_i = 0x0001&valid_a; |
| 29 valid_a_tmp = valid_a_tmp_o; |

30 if

~c1

c1

| 31 a_tmp = 0x7FFF&a; |

32 $

33 L

34 if

~c2

c2

35 $

| 34 b_tmp = 0x7FFF&b |

35 end of if

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.17

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

30 if

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;

30 if

~c1

c1

31 a_tmp = 0x7FFF&a;

32 $

33 L

34 if

~c2

35 $

c2

34 b_tmp = 0x7FFF&b

35 end of if

36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

FIG.18

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

30 if

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

30 if

~c1

41 $

c1

31 a_tmp = 0x7FFF&a;

32 $

33 L

c2

34 if

~c2

35 $

34 b_tmp = 0x7FFF&b

35 end of if

36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.19

LOOP STATEMENT STACK

27 while

BRANCH STATEMENT STACK

30 if

27 while

```
28  valid_a_tmp_i = 0x0001&valid_a;
29  valid_a_tmp   = valid_a_tmp_o;
```

30 if

c1

~c1

```
42  valid_out_i = 0x0001;
43  *valid_out = valid_out_o;
```

41 $

```
31  a_tmp = 0x7FFF&a;
```

32 $

33 L

34 if

~c2

c2

35 $

```
34  b_tmp = 0x7FFF&b
```

35 end of if

```
36  out_i = (a_tmp + b_tmp);
37  *out = out_o;
38  valid_out_i = 0x0001;
39  *valid_out = valid_out_o;
```

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a_ == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.20

LOOP STATEMENT STACK

| 27 while |

BRANCH STATEMENT STACK

| null |

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;

30 if

c1

~c1

41 $

31 a_tmp = 0x7FFF&a;

32 $

33 L

34 if

c2

~c2

35 $

34 b_tmp = 0x7FFF&b

35 end of if

42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;

36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

45 end of if

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

FIG.21

LOOP STATEMENT STACK

null

BRANCH STATEMENT STACK

null

27 while

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;

30 if

~c1

c1

41 $

42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;

31 a_tmp = 0x7FFF&a;

32 $
33 L

c2

~c2

34 if

35 $

34 b_tmp = 0x7FFF&b

35 end of if

36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

45 end of if

46 end of while

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.22

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;

31 a_tmp = 0x7FFF&a;

34 b_tmp = 0x7FFF&b

36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;

~c1

c1

~c2

c2

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.23

FLAG INSERTION

```
28  valid_a_tmp_i = 0x0001&valid_a;
29  valid_a_tmp  = valid_a_tmp_o;
    flg_valid_a_tmp = 1;
```

```
31  a_tmp = 0x7FFF&a;
```

```
34  b_tmp = 0x7FFF&b
```

```
36  out_i = (a_tmp + b_tmp);
37  *out = out_o;
    flg_out = 1;
38  valid_out_i = 0x0001;
39  *valid_out = valid_out_o;
    flg_valid_out = 1;
```

```
42  valid_out_i = 0x0001;
43  *valid_out = valid_out_o;
    flg_valid_out = 1;
```

~c1   c1   ~c2   c2

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.24

DETERMINATION OF INSERTION POSITIONS
OF REGISTER ASSIGNMENT DESCRIPTIONS

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
flg_valid_a_tmp = 1;

~c1

12

42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
flg_valid_out = 1;

c1

31 a_tmp = 0x7FFF&a;

12

12

~c2

c2

34 b_tmp = 0x7FFF&b

36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
flg_out = 1;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
flg_valid_out = 1;

12

valid_a_tmp_o = valid_a_tmp_i;
if (flg_valid_a_tmp==1) valid_a_tmp = valid_a_tmp_o;
out_o   = out_i;
if (flg_out==1) *out = out_o;
valid_out_o = valid_out_i;
if (flg_valid_out==1) *valid_out = valid_out_o;

FIG.25

```c
1  #include <stdio.h>
2  void pipeline(unsigned short valid_a,
3                unsigned short valid_b,
4                unsigned short a,
5                unsigned short b,
6                unsigned short *out,
7                unsigned short *valid_out);
8  main() {
9    unsigned short valid_a, valid_b,
10                  a, b, *out, *valid_out;
11   *out       = 0x0000;
12   *valid_out = 0x0000;
13   pipeline(valid_a, valid_b, a, b, out, valid_out);
14 }

15 void pipeline(unsigned short valid_a, unsigned short valid_b,
16               unsigned short a, unsigned short b,
17               unsigned short *out, unsigned short *valid_out) {
18   unsigned short valid_a_tmp       = 0x0000;
19   unsigned short a_tmp             = 0x0000;
20   unsigned short b_tmp             = 0x0000;
     /* Added variables */
21   unsigned short valid_a_tmp_i;
22   unsigned short valid_a_tmp_o     = 0x0000;
23   unsigned short valid_out_i;
24   unsigned short valid_out_o       = 0x0000;
25   unsigned short out_i;
26   unsigned short out_o             = 0x0000;
27   unsigned short flg_valid_a_tmp = 0x0000;
28   unsigned short flg_valid_out = 0x0000;
29   unsigned short flg_out           = 0x0000;
```

FIG.26

```
30   while (1) {
       /* valid_a_tmp = $ valid_a; */
31     valid_a_tmp_i = 0x0001&valid_a;        /* Refined */
32     valid_a_tmp   = valid_a_tmp_o;         /* Refined */
33     flg_valid_a_tmp = 1;
34     if ((0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)) {
35       a_tmp = 0x7FFF&a;
         /* $ */
         /* BEGIN : Register Assignment */
36       valid_a_tmp_o   = valid_a_tmp_i;
37       if (flg_value_a_tmp == 1) valid_a_tmp     = valid_a_tmp_o;
38       out_o          = out_i;
39       if (flg_out==1) *out = out_o;
40       valid_out_o = valid_out_i;
41       if (flg_valid_out==1) *valid_out = valid_out_o;
         /* END   : Register Assignment */
42     L :
```

FIG.28

GRAPH TRANSFORMATION

[ANY OF LOOP START/END NODE,
CONDITIONAL BRANCH START/END NODE,
AND LABEL BRANCH NODE]

FIG.27

```
43   if (0x0001&valid_b == 0x0001) b_tmp = 0x7FFF&b;
44   else {
         /* $ */
         /* BEGIN : Register Assignment */
45       valid_a_tmp_o  = valid_a_tmp_i;
46       valid_a_tmp    = valid_a_tmp_o;
47       out_o      = out_i;
48       if (flg_out==1) *out = out_o;
49       valid_out_o = valid_out_i;
50       if (flg_valid_out==1) *valid_out = valid_out_o;
         /* END : Register Assignment */
51       goto L;
52   }
     /* *out = $ (a_tmp + b_tmp); */
53   out_i    = a_tmp + b_tmp;        /* Refined */
54   *out     = out_o;               /* Refined */
55   flg_out      = 1;               /* Added */
     /* *valid_out = $ 0x0001; */
56   valid_out_i  = 0x0001;          /* Refined */
57   *valid_out    = valid_out_o;    /* Refined */
58   flg_valid_out  = 1;             /* Added */
59   } else {
         /* $ */
         /* BEGIN : Register Assignment */
60       valid_a_tmp_o  = valid_a_tmp_i;
61       valid_a_tmp    = valid_a_tmp_o;
62       out_o    = out_i;
63       if (flg_out==1) *out = out_o;
64       valid_out_o = valid_out_i;
65       if (flg_valid_out==1) *valid_out = valid_out_o;
         /* END : Register Assignment */
         /* *valid_out = $ 0x0000; */
66       valid_out_i = 0x0000;           /* Refined */
67       *valid_out    = valid_out_o;    /* Refined */
68       flg_valid_out  = 1;            /* Added */
69   }
70   }
71   }
```
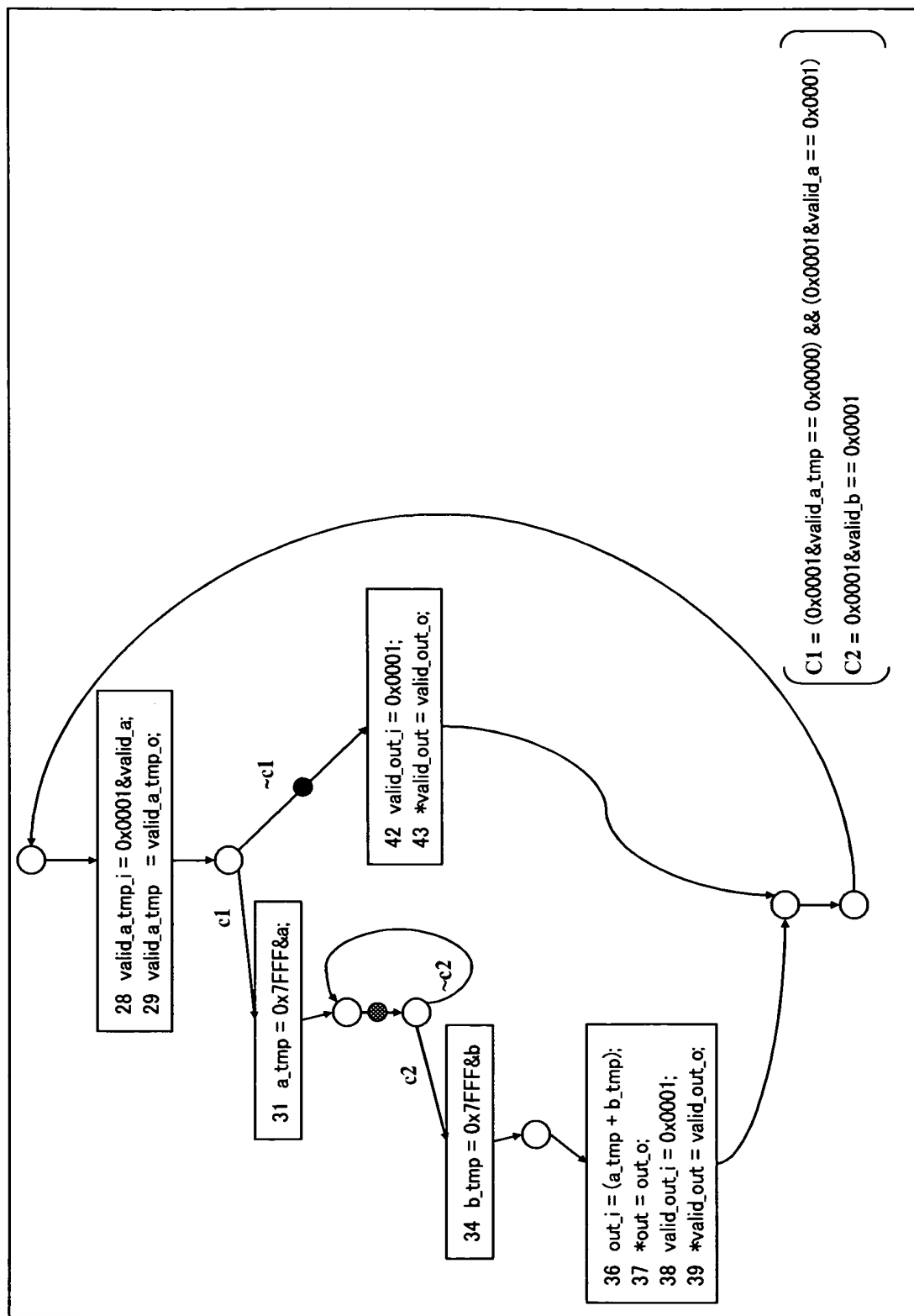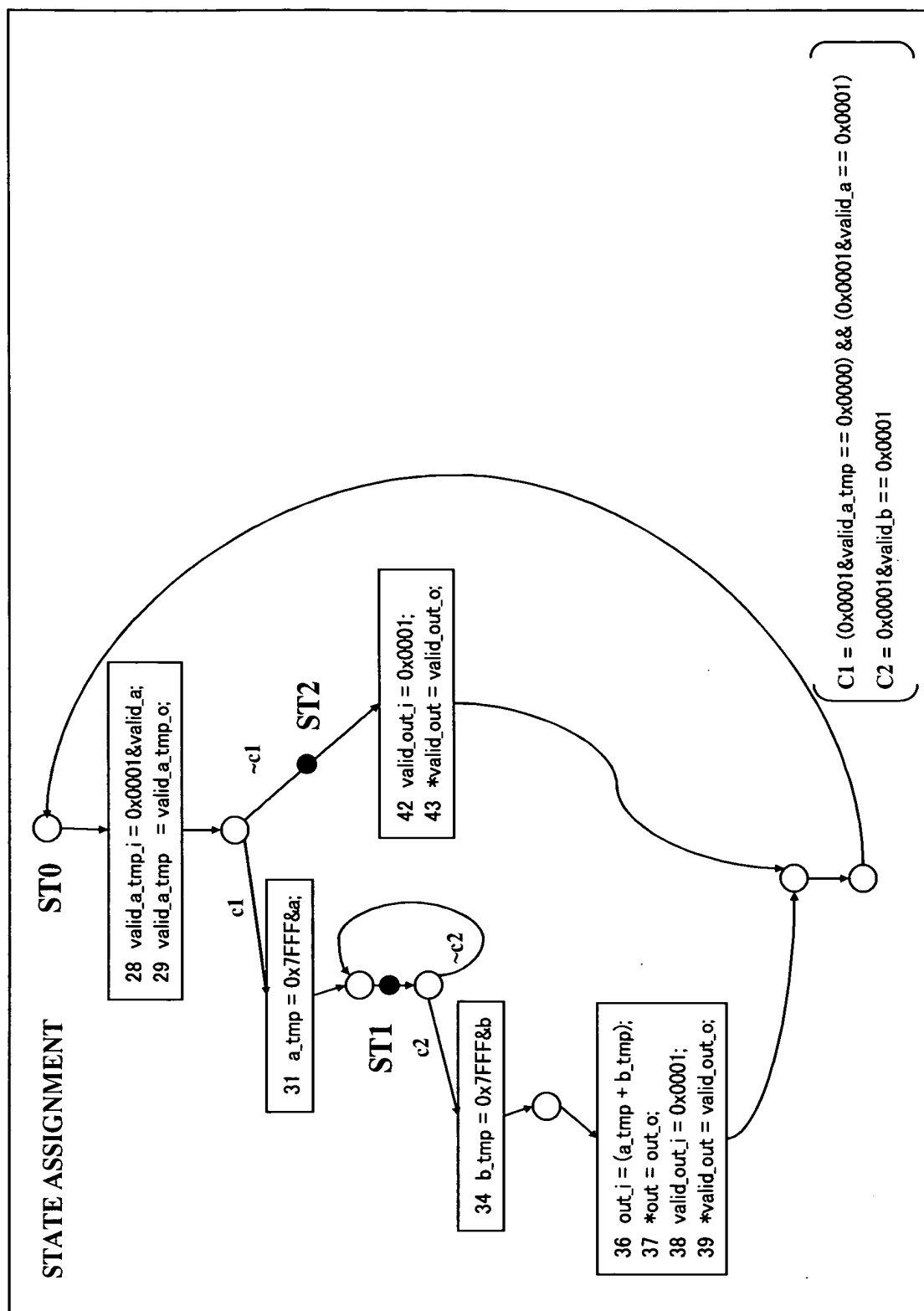
FIG.29

GRAPH TRANSFORMATION

CLOCK BOUNDARIES OF
PRECEDING STAGE
DO NOT CONTAIN
THOSE OF OUTPUT SIDES

THAT NODE AMONG LOOP START/END NODE, CONDITIONAL BRANCH
START/END NODE AND LABEL BRANCH NODE WHICH HAS PLURALITY
OF OUTPUT SIDES, NEITHER INPUT SIGNAL NOR OUTPUT SIGNAL
BEING CONTAINED IN CONDITIONS AFFIXED TO OUTPUT SIDES,
AND AT LEAST TWO OUTPUT SIDES OF WHICH HAVE
CLOCK BOUNDARIES AFFIXED THERETO

FIG.30

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

~c1

42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;

c1

31 a_tmp = 0x7FFF&a;

~c2

c2

34 b_tmp = 0x7FFF&b

36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

C1 = (0x0001&valid_a_tmp = = 0x0000) && (0x0001&valid_a = = 0x0001)
C2 = 0x0001&valid_b = = 0x0001

FIG.31

STATE ASSIGNMENT

ST0

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

~c1

ST2

42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;

c1

31 a_tmp = 0x7FFF&a;

ST1

c2

~c2

34 b_tmp = 0x7FFF&b

36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

FIG.32

```
1   void foo(unsigned short in,
2               unsigned short cond,
3               unsigned short *out) {
4   unsigned short a, b, c, d, e;
5   while(1) {
6     a= 0;
7     b = 0;
8     a = in;
9     b = a + 1;
10    if (cond) {
11      c = b + 1;
12      d = c + 1;
13      e  = d + 2;
14      if (d > 6) c--;
15      else      c++;
16      c = c + 5;
17      $
18      *out = c;
19    } else {
20      $
21      *out = 1;
22    }
23    $
24  }
```

FIG.33

CFG

```
6   a= 0;
7   b = 0;
8   a = in;
9   b = a + 1;
```

c1   !c1

```
11  c = b + 1;
12  d = c + 1;
13  e = d + 2;
```

21 *out = 1;

c2   !c2

14 c--   15 c++

16 c = c+ 5;

18 *out = c;

$$\begin{pmatrix} c1 = cond; \\ c2 = (d>6); \end{pmatrix}$$

FIG.34

STATE ASSIGNMENT

ONLY ONE INPUT SIDE TO START NODE EXISTS,
AND CLOCK BOUNDARY IS AFFIXED THERETO

ST0

6 a= 0;
7 b = 0;
8 a = in;
9 b = a + 1;

11 c = b + 1;
12 d = c + 1;
13 e = d + 2;

!c1   ST2

21 *out = 1;

15 c++

14 c--

16 c = c + 5;

18 *out = c;

ST1

c1 = cond;
c2 = (d>6);

FIG.35

CREATION OF VARIABLE TABLE

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[6], def[8], use@b[9]
	b	def[7], def[9], pred(c1)[use@c[11]]
	c	pred(c1)[def[11], use@d[12], pred(c2)[def[14]use, def[16]use]]
	d	pred(c1)[def[12], use@pred(c2)]
	e	pred(c1)[def[13]]
ARGUMENT	cond	use@pred(c1)
	out	

ST0

ST2

ST1

!c1

c1

!c2

c2

c2

6 a=0;
7 b=0;
8 a = in;
9 b=a+1;

11 c=b+1;
12 d=c+1;
13 e=d+2;

14 c--

15 c++

16c= c+5;

18 *out = c;

21 *out = 1;

c1 = cond;
c2 = (d>6);

FIG. 36

CREATION OF VARIABLE TABLE

ST0

ST2

!c1

!c1

21 *out = 1;

6 a = 0;
7 b = 0;
8 a = in;
9 b = a + 1;

c1

11 c=b + 1;
12 d=c + 1;
13 e=d+2;

!c2

15c++;

c2

14 c--

16c=c +5;

ST1

18 *out = c;

c1 = cond;
c2 = (d>6);

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[6], def[8], use@b[9]
	b	def[7], def[9], pred(c1)use@c[11]
	c	pred(c1)def[11], use@d[12], pred(!c2)[def[14]use, def[16]use]}
	d	pred(c1)def[12], use@pred(!c2)}
	e	pred(c1)def[13]}
ARGUMENT	cond	use@pred(c1)
	out	

FIG.37

CREATION OF VARIABLE TABLE

STATE TRANSITION		ST0->ST2
LOCAL VARIABLE	a	def[6], def[8], use@b[9]
	b	def[7], def[9]
	c	
	d	
	e	
ARGUMENT	cond	use@pred(!c1)
	out	

FIG.38

CREATION OF VARIABLE TABLE

6 a = 0;
7 b = 0;
8 a = in;
9 b = a + 1;

11 c = b + 1;
12 d = c + 1;
13 e = d + 2;

14 c--

15 c++

16 c = c + 5;

18*out = c;

21 *out = 1;

c1 = cond;
c2 = (d>6);

ST2

ST1

ST0

STATE TRANSITION		ST1->ST0
LOCAL VARIABLE	a	
	b	
	c	use@out[17]
	d	
	e	
ARGUMENT	cond	
	out	def[18]

FIG.39

CREATION OF VARIABLE TABLE

6 a = 0;
7 b = 0;
8 a = in;
9 b = a + 1;

ST2

!c1

21*out = 1;

c1

11 c = b + 1;
12 d = c + 1;
13 e = d + 2;

!c2

15 c++

c2

14 c--

16 c = c + 5;

ST1

18 *out = c;

ST0

c1 = cond;
c2 = (d>6);

STATE TRANSITION	ST2->ST0
LOCAL VARIABLE	a
	b
	c
	d
	e
ARGUMENT	cond
	out def[21]

FIG.40

VARIABLE TABLE CREATION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	a	def[6], def[8], use@b[9]	def[6], def[8], use@b[9]		
	b	def[7], def[9], pred(c1)[use@c[11]]	def[7], def[9]		
	c	pred(c1)[def[11], use@d[12], pred(c2)[def[14]use, def[16]use]]		use@out[17]	
	d	pred(c1)[def[12], use@pred(c2)]			
	e	pred(c1)[def[13]]			
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out			def[18]	def[21]

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[6], def[8], use@b[9]
	b	def[7], def[9], pred(c1)[use@c[11]]
	c	pred(c1)[def[11], use@d[12], pred(!c2)[def[14]use, def[16]use]]
	d	pred(c1)[def[12], use@pred(!c2)]
	e	pred(c1)[def[13]]
ARGUMENT	cond	use@pred(c1)
	out	

def[n]	: EXPRESSING THAT VARIABLE IS DEFINED AT LINE "n"
use@var[m]	: EXPRESSING THAT VARIABLE IS USED FOR ASSIGNMENT TO VARIABLE "var" AT LINE "m"
pred(cond){...}	: EXPRESSING THAT {...} IS PERFORMED IN CASE WHERE BRANCH OF CONDITION "cond" HAS HELD
def[l]use	: EXPRESSING THAT VARIABLE IS USED FOR ASSIGNMENT TO VARIABLE ITSELF AT LINE 1
use@pred(cond)	: EXPRESSING THAT VARIABLE IS USED IN CONDITION "cond"

FIG.41

REDUNDANT STATEMENT DELETION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	a	def[6], def[8], use@b[9]	def[6], def[8], use@b[9]		
	b	def[7], def[9], pred(c1)[use@c[11]]	def[7], def[9]		
	c	pred(c1){def[11], use@d[12], pred(c2)[def[14]use, def[16]use]}		use@out[17]	
	d	pred(c1){def[12], use@pred(c2)}			
	e	pred(c1){def[13]}			def[21]
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out			def[18]	

STATE TRANSITION		ST0->ST1			
LOCAL VARIABLE	a	def[6], def[8], use@b[9]			
	b	def[7], def[9], pred(c1)[use@c[11]]			
	c	pred(c1){def[11], use@d[12], pred(!c2)[def[14]use, def[16]use]}			
	d	pred(c1){def[12], use@pred(!c2)}			
	e	pred(c1){def[13]}			
ARGUMENT	cond	use@pred(c1)			
	out				

10/531287

FIG. 42

REDUNDANT STATEMENT DELETION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	a	def[8], use@b[9]	def[8], use@b[9]		
	b	def[9], pred(c1)[use@c[11]]	def[9]	use@out[17]	
	c	pred(c1)[def[11], use@d[12], pred(!c2)[def[14]use, def[16]use]]			
	d	pred(c1)[def[12], use@pred(c2)]			
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)	def[18]	def[21]
	out				

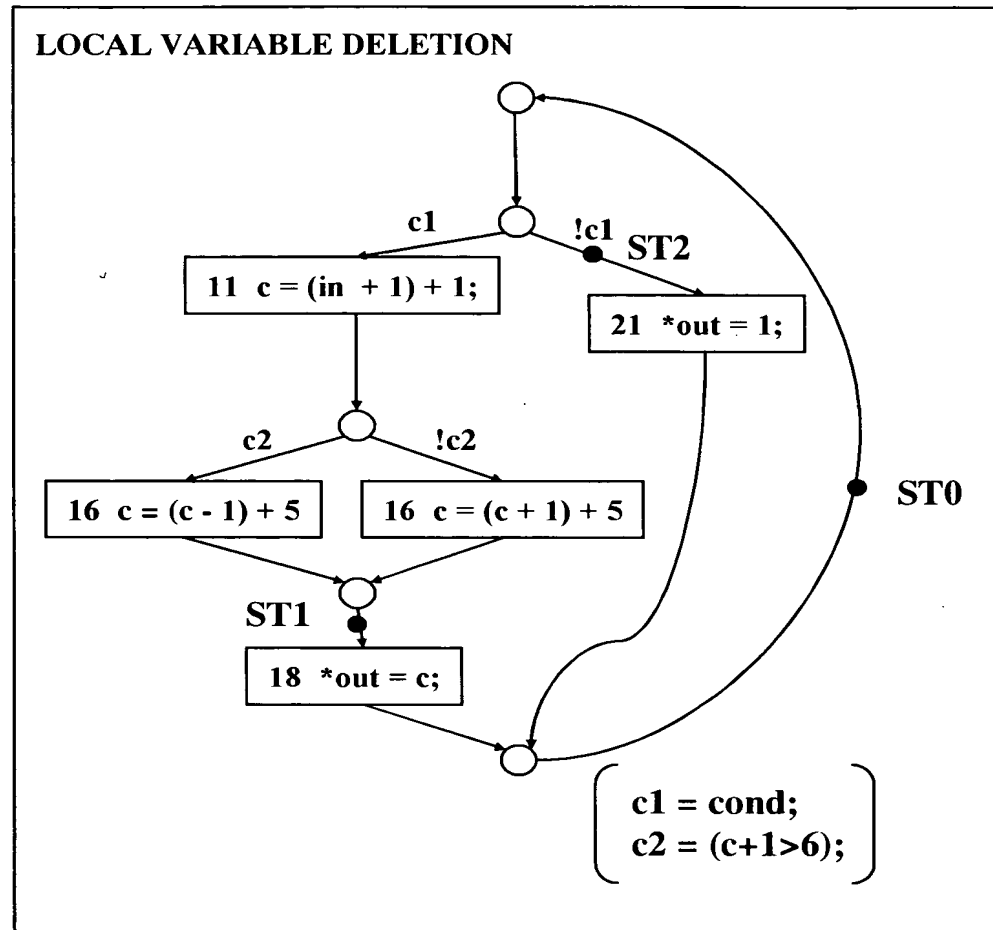| STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[8], use@b[9]
	b	def[9], pred(c1)[use@c[11]]
	c	pred(c1)[def[11], use@d[12], pred(!c2)[def[14]use, def[16]use]]
	d	pred(c1)[def[12], use@pred(!c2)]
ARGUMENT	cond	use@pred(c1)
	out	

FIG.43

REDUNDANT STATEMENT DELETION

8   a = in;
9   b = a + 1;

c1   !c1   ST2

11 c = b + 1;
12 d = c + 1;

21 *out = 1;

c2   !c2

14 c--   15 c++

STO

16 c = c+ 5;

ST1

18 *out = c;

c1 = cond;
c2 = (d>6);

FIG.44

LOCAL VARIABLE DELETION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	a	def[8], use@b[9]	def[8], use@b[9]		
	b	def[9], pred(c1)[use@c[11]]	def[9]		
	c	pred(c1)[def[11], use@d[12], pred(c2)[def[14]use, def[16]use]]		use@out[17]	
	d	pred(c1)[def[12], use@pred(c2)]			
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out			def[18]	def[21]

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[8], use@b[9]
	b	def[9], pred(c1)[use@c[11]]
	c	pred(c1)[def[11], use@d[12], pred(!c2)[def[14]use, def[16]use]]
	d	pred(c1)[def[12], use@pred(!c2)]
ARGUMENT	cond	use@pred(c1)
	out	

FIG.45

LOCAL VARIABLE DELETION

c1

11 c = (in + 1) + 1;

!c1 ST2

21 *out = 1;

c2   !c2

16 c = (c - 1) + 5   16 c = (c + 1) + 5

ST1

18 *out = c;

ST0

c1 = cond;
c2 = (c+1>6);

FIG.46

AFTER UPDATING

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	c	pred(c1){def[11], pred(c2){def[16]use}}			
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)	use@out[17]	
	out			def[18]	def[21]

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	c	pred(c1){def[11], pred(!c2){def[16]use}}
ARGUMENT	cond	use@pred(c1)
	out	

FIG.47

STATE TRANSITION			ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	c		pred(c1)[def[11], pred(c2)[def[16]use]]	retain	retain	retain
ARGUMENT		cond	use@pred(c1)	use@pred(!c1)		
		out	retain	retain	def[18]	def[21]

STATE TRANSITION			ST0->ST1			
LOCAL VARIABLE	c		pred(c1)[def[11], pred(!c2)[def[16]use]]			
ARGUMENT		cond	use@pred(c1)			
		out	retain			

FIG.48

SIMPLIFICATION OF
ARITHMETIC FORMULA

c1

11 c = in + 2;

!c1 ST2

21 *out = 1;

c2   !c2

16 c = c + 4   16 c = c + 6

ST1

18 *out = c;

ST0

$$c1 = cond;$$
$$c2 = (c>5);$$

FIG.49

CFG AFTER OPTIMIZATION

```
DELETED VARIABLE
b_tmp
```

ST0

```
28  valid_a_tmp_i = 0x0001&valid_a;
29  valid_a_tmp   = valid_a_tmp_o;
```

~c1   ● ST2

```
42  valid_out_i = 0x0001;
43  *valid_out = valid_out_o;
```

c1

```
31  a_tmp = 0x7FFF&a;
```

ST1 ● ~c2

c2

```
36  out_i = (a_tmp + 0x7FFF&b);
37  *out = out_o;
38  valid_out_i = 0x0001;
39  *valid_out = valid_out_o;
```

```
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001
```

FIG.50

AFTER OPTIMIZATION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST1
LOCAL VARIABLE	valid_a_tmp	def[29], use@pred(c1)	def[29], use@pred(!c1)	pred(c2)[def[29], use@pred(c1)]
	valid_a_tmp_l	def[28]	def[28]	pred(c2)[def[28]]
	a_tmp	pred(c1)[def[31]]		pred(c2)[pred(c1)[def[31]]]
	out_l			pred(c2)[def[36]]
	valid_out_i			pred(c2)[def[38]]
ARGUMENT	a	pred(c1)[use@a_tmp[31]]		pred(c2)[pred(c1)[use@a_tmp[31]]]
	b			pred(c2)[use@out_i[36]]
	valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)	pred(c2)[use@valid_a_tmp_i[28], use@pred(c1)]
	valid_b			use@pred(c2)
	valid_out			pred(c2)[def[39]]
	out			pred(c2)[def[37]]
	valid_a_tmp_o	use@valid_a_tmp[29]	use@valid_a_tmp[29]	pred(c2)[use@valid_a_tmp[29]]
	valid_out_o			pred(c2)[use@valid_out_o[39]]
	out_o			pred(c2)[use@out[37]]

STATE TRANSITION		ST1->ST2	ST2->ST1	ST2->ST2
LOCAL VARIABLE	valid_a_tmp	pred(c2)[def[29], use@pred(!c1)]	def[29], use@pred(c1)	def[29], use@pred(!c1)
	valid_a_tmp_l	pred(c2)[def[28]]	def[28]	def[28]
	a_tmp	use@pred(c2)	pred(c1)[def[31]]	
	out_l	pred(c2)[def[36]]	def[42]	def[42]
	valid_out_i	pred(c2)[def[38]]		
ARGUMENT	a	pred(c2)[use@out_i[36]]	pred(c1)[use@a_tmp[31]]	
	b	pred(c2)[use@valid_a_tmp_i[28], use@pred(c1)]		
	valid_a	use@pred(c2)	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(c1)
	valid_b	pred(c2)[def[39]]		
	valid_out	pred(c2)[def[37]]		
	out	pred(c2)[use@valid_a_tmp[29]]	def[43]	def[43]
	valid_a_tmp_o	pred(c2)[use@valid_out_o[39]]		
	valid_out_o	pred(c2)[use@out[37]]	use@valid_out[43]	use@valid_out[43]
	out_o			

FIG.51

HIGHEST-LEVEL NODE

!cond_0

cond_0

cond_1

!cond_1

use@var_1[j]

!cond_2

cond_2

def[k]

retain

ENTER
pred(cond_0){pred(cond_1){pred(!cond_2){retain}}}
INTO VARIABLE TABLE

PARTIAL TREE DELETION, AND "RETAIN" IDENTIFICATION

HIGHEST-LEVEL NODE

!cond_0

cond_0

cond_1

!cond_1

use@var_1[j]

!cond_2

cond_2

def[k]

!cond_3

cond_3

def[s]

PARTIAL TREE WHICH IS LOWER IN LEVEL THAN "def"

FIG. 52

AFTER EXECUTION OF "RETAIN" ANALYSIS

STATE TRANSITION		ST0→ST1	ST0→ST2	ST1→ST1	ST1→ST1
LOCAL VARIABLE	valid_a_tmp	def[29], use@pred(!c1)	def[29], use@pred(!c1)	pred(c2)[def[29], use@pred(c1)]	pred(!c2)[retain]
	valid_a_tmp_I	def[28]	def[28]	pred(c2)[def[28]]	pred(!c2)[retain]
	a_tmp	pred(c1)[def[31]]	pred(!c1)[retain]	pred(c2)[pred(c1)[def[31]]]	pred(!c2)[retain]
	out_I	retain	retain	pred(c2)[def[36]]	pred(!c2)[retain]
	valid_out_i	retain	retain	pred(c2)[def[38]]	pred(!c2)[retain]
ARGUMENT	a	pred(c1)[use@a_tmp[31]]		pred(c2)[pred(c1)[use@a_tmp[31]]]	
	b			pred(c2)[use@out_i[36]]	
	valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)	pred(c2)[use@valid_a_tmp_i[28], use@pred(c1)]	
	valid_b			use@pred(c2)	use@pred(!c2)
	valid_out	retain	retain	pred(c2)[def[39]]	pred(!c2)[retain]
	out	retain	retain	pred(c2)[def[37]]	pred(!c2)[retain]
	valid_a_tmp_o	use@valid_a_tmp[29]	use@valid_a_tmp[29]	pred(c2)[use@valid_a_tmp[29]]	
	valid_out_o			pred(c2)[use@valid_out_o[39]]	
	out_o			pred(c2)[use@out[37]]	

STATE TRANSITION		ST1→ST2	ST2→ST1	ST2→ST2
LOCAL VARIABLE	valid_a_tmp	pred(c2)[def[29], use@pred(!c1)]	def[29], use@pred(c1)	def[29], use@pred(!c1)
	valid_a_tmp_I	pred(c2)[def[28]]	def[28]	def[28]
	a_tmp	pred(!c1)[retain]	pred(c1)[def[31]]	pred(!c1)[retain]
	out_I	pred(c2)[def[36]]	retain	retain
	valid_out_i	pred(c2)[def[38]]	def[42]	def[42]
ARGUMENT	a	pred(c2)[use@out_i[36]]	pred(c1)[use@a_tmp[31]]	pred(c1)[use@a_tmp[31]]
	b			
	valid_a	pred(c2)[use@valid_a_tmp_i[28], use@pred(!c1)]	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)
	valid_b	use@pred(c2)		
	valid_out	pred(c2)[def[39]]	def[43]	def[43]
	out	pred(c2)[def[37]]	retain	retain
	valid_a_tmp_o	pred(c2)[use@valid_a_tmp[29]]		
	valid_out_o	pred(c2)[use@valid_out_o[39]]	use@valid_out[43]	use@valid_out[43]
	out_o	pred(c2)[use@out[37]]		

FIG.53

INFORMATION ACQUISITION FROM VARIABLE TABLE WHICH IS "RETAIN" ANALYSIS RESULT

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST1	ST1->ST1
LOCAL VARIABLE	valid_a_tmp	def[29], use@pred(lc1)	def[29], use@pred(lc1)	pred(c2)[def[29], use@pred(c1)]	pred(!c2)[valid_a_tmp = valid_a_tmp_o;]
	valid_a_tmp_i	def[28]	def[28]	pred(c2)[def[28]]	pred(!c2)[valid_a_tmp_i = valid_a_tmp_o;]
	a_tmp	pred(c1)[def[31]]	pred(!c1)[nxt_a_tmp = tmp;]	pred(c2)[pred(c1)[def[31]]]	pred(!c2)[nxt_a_tmp = a_tmp;]
	out_i	out_i = out_o;	out_i = out_o;	pred(c2)[def[36]]	pred(!c2)[out_i = out_o;]
	valid_out_i	valid_out_i = valid_out_o;	valid_out_i = valid_out_o;	pred(c2)[def[38]]	pred(!c2)[valid_out_i = valid_out_o;]
ARGUMENT	a	pred(c1)[use@a_tmp[31]]		pred(c2)[pred(c1)[use@a_tmp[31]]]	
	b			pred(c2)[use@out_i[36]]	
	valid_a	use@valid_a_tmp_i[28], use@pred(lc1)	use@valid_a_tmp_i[28], use@pred(lc1)	pred(c2)[use@valid_a_tmp_i[28], use@pred(lc1)]	
	valid_b			use@pred(c2)	use@pred(lc2)
	valid_out	valid_out = valid_out_o;	valid_out = valid_out_o;	pred(c2)[def[39]]	pred(!c2)[valid_out = valid_out_o;]
	out	out = out_o;	out = out_o;	pred(c2)[def[37]]	pred(!c2)[out = out_o;]
	valid_a_tmp_o	use@valid_a_tmp[29]	use@valid_a_tmp[29]	pred(c2)[use@valid_a_tmp[29]]	
	valid_out_o			pred(c2)[use@valid_out_o[39]]	
	out_o			pred(c2)[use@out[37]]	

STATE TRANSITION		ST1->ST2	ST2->ST1	ST2->ST2	
LOCAL VARIABLE	valid_a_tmp	pred(c2)[def[29], use@pred(lc1)]	def[29], use@pred(lc1)	def[29], use@pred(lc1)	
	valid_a_tmp_i	pred(c2)[def[28]]	def[28]	def[28]	
	a_tmp	pred(!c1)[nxt_a_tmp = a_tmp;]	pred(c1)[def[31]]	pred(!c1)[nxt_a_tmp = a_tmp;]	
	out_i	pred(c2)[def[36]]	out_i = out_o;	out_i = out_o;	
	valid_out_i	pred(c2)[def[38]]	def[42]	def[42]	
ARGUMENT	a		pred(c1)[use@a_tmp[31]]	pred(c1)[use@a_tmp[31]]	
	b	pred(c2)[use@out_i[36]]			
	valid_a	pred(c2)[use@valid_a_tmp_i[28], use@pred(lc1)]	use@valid_a_tmp_i[28], use@pred(lc1)	use@valid_a_tmp_i[28], use@pred(lc1)	
	valid_b	use@pred(c2)			
	valid_out	pred(c2)[def[39]]	def[43]	def[43]	
	out	pred(c2)[def[37]]	out = out_o;	out = out_o;	
	valid_a_tmp_o	pred(c2)[use@valid_a_tmp[29]]			
	valid_out_o	pred(c2)[use@valid_out_o[39]]	use@valid_out[43]	use@valid_out[43]	
	out_o	pred(c2)[use@out[37]]			

FIG.54

START STATE : ST0

CODES ARE GENERATED FROM
EACH STATE BY DFS.
ESPECIALLY, STATE VARIABLES
HAVE CODES GENERATED IN FORMS
OF nxt_state = ST0, ETC.

```
ST0 : begin
        valid_a_tmp_i = valid_a;
        valid_a_tmp   =
valid_a_tmp_o;
        if (c1) begin          }ST1への分岐
              nxt_a_tmp = a;
              nxt_state = ST1;
        end
        else begin             }ST2への分岐
              nxt_state = ST2;
        end

end
```

ST0

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

~c1

ST2

42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;

c1

31 a_tmp = 0x7FFF&a;

ST1

c2   ~c2

36 out_i = (a_tmp + 0x7FFF&b);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

FIG.55

START STATE : ST0

	ST0->ST1	ST0->ST2
valid_a_tmp	def[29], use@pred(c1)	def[29], use@pred(!c1)
valid_a_tmp_I	def[28]	def[28]
a_tmp	pred(c1)[def[31]]	pred(!c1)[nxt_a_tmp = tmp;]
out_I	out_i = out_o;	out_i = out_o;
valid_out_i	valid_out_i = valid_out_o;	valid_out_i = valid_out_o;
a	pred(c1)[use@a_tmp[31]]	
b		
valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)
valid_b		
valid_out	valid_out = valid_out_o;	valid_out = valid_out_o;
out	out = out_o;	out = out_o;
valid_a_tmp_o	use@valid_a_tmp[29]	use@valid_a_tmp[29]
valid_out_o		
out_o		

DEPENDING ONLY UPON
START STATE

INSERT

```
ST0 : begin
    out_i = out_o;
    valid_out_i = valid_out_o;
    valid_out = valid_out_o;
    out = out_o;
    valid_a_tmp_i = valid_a;
    valid_a_tmp = valid_a_tmp_o;
    if (c1) begin
        nxt_a_tmp = a;
        nxt_state = ST1;
    end
    else begin
        nxt_a_tmp = a_tmp;
        nxt_state = ST2;
    end
end
```

USE VARIABLE NAME
BEARING nxt_

INSERT

DEPENDING UPON ARRIVAL STATE
AND BRANCH CONDITION pred(!c1)

FIG.56

FIG.57

START STATE : ST1

	ST1→ST1	ST1→ST1	ST1→ST2
valid_a_tmp		pred(c2)[valid_a_tmp = valid_a_tmp_o]	pred(c2)[def[29], use@pred(!c1)]
valid_a_tmp_i		pred(c2)[valid_a_tmp_i = valid_a_tmp_o]	pred(c2)[def[28]]
a_tmp	pred(c2)[pred(c1)[def[31]]]	pred(c2)[nxt_a_tmp = tmp]	pred(!c1)[nxt_a_tmp = a_tmp]
out_i	pred(c2)[def[36]]	pred(c2)[out_i = out_o]	pred(c2)[def[36]]
valid_out_i	pred(c2)[def[38]]	pred(!c2)[valid_out_i = valid_out_o]	pred(c2)[def[38]]
a	pred(c2)[pred(c1)[use@a_tmp[31]]]		pred(c2)[use@out_i[36]]
b	pred(c2)[use@out_i[36]]		pred(c2)[use@valid_a_tmp_i[28], use@pred(c1)]
valid_a	pred(c2)[use@valid_a_tmp_i[28], use@pred(c1)]		use@pred(c2)
valid_b	use@pred(c2)		pred(c2)[def[39]]
valid_out	pred(c2)[def[39]]	use@pred(!c2)	pred(c2)[def[37]]
out	pred(c2)[def[37]]	pred(!c2)[valid_out = valid_out_o]	pred(c2)[use@valid_a_tmp[29]]
valid_a_tmp_o	pred(c2)[use@valid_a_tmp[29]]	pred(!c2)[out = out_o]	pred(c2)[use@valid_out_o[39]]
valid_out_o	pred(c2)[use@valid_out_o[39]]		pred(c2)[use@out[37]]
out_o	pred(c2)[use@out[37]]		

DEPENDING UPON ARRIVAL STATE AND BRANCH CONDITION pred(!c1)

DEPENDING UPON ARRIVAL STATE AND BRANCH CONDITION pred(!c2)

INSERT

INSERT

```
ST1 : begin
  if (c2) begin
    out_i = a_tmp + b;
    out  = out_o;
    valid_put_tmp_i= 1'b1;
    valid_out = valid_out_o;
    valid_a_tmp_i= valid_a;
    valid_a_tmp = valid_a_tmp_o;
    if (c1) begin
      nxt_a_tmp = a;
      nxt_state = ST1;
    end
    else begin
      nxt_a_tmp = a_tmp;
      nxt_state = ST2;
    end
  end
  else begin
    valid_a_tmp = valid_a_tmp_o;
    valid_a_tmp_i = valid_a_tmp_o;
    nxt_a_tmp = a_tmp;
    out_i = out_o;
    valid_out_i= valid_out_o;
    valid_out = valid_out_o;
    out = out_o;
    nxt_state = ST1;
  end
end
```

FIG.58

START STATE : ST1

ST0

28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;

~c1

c1

ST2

31 a_tmp= 0x7FFF&a;

ST1

c2

~c2

42 valid_out.i   = 0x0001;
43*valid_out   = valid_out_o;

36 out.i = (a_tmp + 0x7FFF&b);
37 *out = out_o;
38 valid_out.i = 0x0001;
39 *valid_out = valid_out_o;

ST2 : begin
    valid_out_I = 1'b1;
    valid_out = valid_out_o;
    valid_a_tmp_I = valid_a;
    valid_a_tmp = valid_a_tmp_o;
    if (c1) begin
        nxt_a_tmp = a;
        nxt_state = ST1;
    end
    else begin
        nxt_state = ST2;
    end
end

FIG.59

START STATE : ST2

	ST2->ST1	ST2->ST2
valid_a_tmp	def[29], use@pred(c1)	def[29], use@pred(!c1)
valid_a_tmp_l	def[28]	def[28]
a_tmp	pred(c1)|def[31]]	pred(!c1)|nxt_a_tmp = a_tmp;]
out_l	out_i = out_o;	out_i = out_o;
valid_out_i	def[42]	def[42]
a	pred(c1)|use@a_tmp[31]]	
b		
valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)
valid_b		
valid_out	def[43]	def[43]
out	out = out_o;	out = out_o;
valid_a_tmp_o		
valid_out_o	use@valid_out[43]	use@valid_out[43]
out_o		

DEPENDING ONLY UPON START STATE

DEPENDING UPON ARRIVAL STATE AND BRANCH CONDITION pred (!c1)

INSERT

```
ST2 : begin
    out_i = out_o;
    out = out_o;
    valid_out_I = 1'b1;
    valid_out = valid_out_o;
    valid_a_tmp_I = valid_a;
    valid_a_tmp = valid_a_tmp_o;
    if (c1) begin
        nxt_a_tmp = a;
        nxt_state = ST1;
    end
    else begin
        nxt_a_tmp = a_tmp;
        nxt_state = ST2;
    end
end
```

FIG.60

```
1  module PipeLine(clk, reset_n,
2              valid_a, valid_b, a, b,
3              out, valid_out);
       // System clock and reset
4  input clk;
5  input reset_n;
       // PipeLine input signals
6  input valid_a;
7  input valid_b;
8  input [14:0] a;
9  input [14:0] b;
       // PipeLine output signals
10 output valid_out;
11 reg valid_out;
12 output [15:0] out;
13 reg [15:0] out;
```

```
       // PipeLine internal signals
14 reg valid_a_tmp;
15 reg valid_a_tmp_i;
16 reg valid_a_tmp_o;
17 reg [14:0] a_tmp;
18 reg [14:0] nxt_a_tmp;
19 reg valid_out_i;
20 reg valid_out_o;
21 reg [15:0] out_i;
22 reg [15:0] out_o;
       // State registers
23 reg [1:0] state, nxt_state;
24 parameter ST0=2'b00,
25              ST1=2'b01,
26              ST2=2'b10;
       // Blanch conditions
27 wire c1;
28 wire c2;
29 assign c1 = !valid_a_tmp&&valid_a;
30 assign c2 = valid_b;
```

FIG.61

```verilog
   // Regsiter assignment statement
31 always @ (posedge clk or negedge reset_n) begin
32   if (!reset_n) begin
33     valid_a_tmp_o <= 1'b0;
34     out_o         <= 17'b00000000000000000;
35   end
36   else begin
37     valid_a_tmp_o <= valid_a_tmp_i;
38     out_o         <= out_i;
39   end
40 end

   // State registers and temporal registers
41 always @ (posedge clk or negedge reset_n) begin
42   if (!reset_n) begin
43     state <= ST0;
44     a_tmp <= 16'b0;
45   end
46   else begin
47     state <= nxt_state;
48     a_tmp <= nxt_a_tmp;
49   end
50 end

   // Mealy finite state machine
51 always @ (state or c1 or c2 or
52           valid_a_tmp_i or valid_a_tmp_o or
53           valid_a_tmp or a_tmp or
54           valid_out_i or valid_out_o or
55           out_i or out_o) begin
56   case(state[1:0])
57   ST0 : begin
58     valid_a_tmp_i = valid_a;
59     valid_a_tmp = valid_a_tmp_o;
60     valid_out_i = valid_out_o;
61     valid_out   = valid_out_o;
62     out_i       = out_o;
63     out         = out_o;
64     if (c1) begin
65       nxt_a_tmp = a;
66       nxt_state = ST1;
67     end
68     else begin
69       nxt_a_tmp = a_tmp;
70       nxt_state = ST2;
71     end
72 end
```

FIG.62

```verilog
72  ST1 : begin
73    if (c2) begin
74      out_i        = a_tmp + b;
75      out          = out_o;
76      valid_out_i = 1'b1;
77      valid_out   = valid_out_o;
78      valid_a_tmp_i = valid_a;
79      valid_a_tmp = valid_a_tmp_o;
80      if (c1) begin
81        nxt_a_tmp = a;
82        nxt_state = ST1;
83      end
84      else begin
85        nxt_a_tmp = a_tmp;
86        nxt_state = ST2;
87      end
88    end
89    else begin
90      nxt_state    = ST1;
91      valid_a_tmp_i = valid_a_tmp_o;
92      valid_a_tmp = valid_a_tmp_o;
93      nxt_a_tmp    = a_tmp;
95      valid_out_i = valid_out_o;
96      valid_out   = valid_out_o;
97      out_i        = out_o;
98      out          = out_o;
99    end
100   end

101  ST2 : begin
102      valid_a_tmp_i  = valid_a;
103      valid_a_tmp = valid_a_tmp_o;
104      valid_out_i = 1'b0;
105      valid_out    = valid_out_o;
106      out_i        = out_o;
107      out          = out_o;
108      if (c1) begin
109        nxt_a_tmp = a;
110        nxt_state = ST1;
111      end
112      else begin
113        nxt_a_tmp = a_tmp;
114        nxt_state = ST2;
115      end
116    end
117    default : begin
118      nxt_state    = ST0;
119      valid_a_tmp_i = valid_a_tmp_o;
120      valid_a_tmp = 1'b0;
121      nxt_a_tmp    = 15'b0;
122      valid_out_i = valid_out_o;
123      valid_out    = 1'b0;
124      out_i        = out_o;
125      out          = out_o;
126    end
127    endcase
128  end
129 endmodule
```